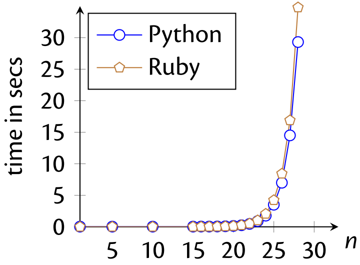
Compilers and Formal Languages

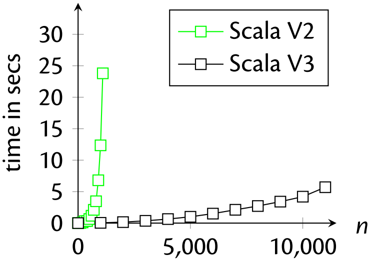
Email:	christian.urban at kcl.ac.uk
Office Hour:	Thurdays 15 – 16
Location:	N7.07 (North Wing, Bush House)
Slides & Progs:	KEATS
Pollev:	https://pollev.com/cfltutoratki576

1 Introduction, Languages	6 While-Language
2 Regular Expressions, Derivatives	7 Compilation, JVM
3 Automata, Regular Languages	8 Compiling Functional Languages
4 Lexing, Tokenising	9 Optimisations
5 Grammars, Parsing	10 LLVM

Let's Implement an Efficient Regular Expression Matcher

Graphs: $a^{?\{n\}} \cdot a^{\{n\}}$ and strings $\underbrace{a \ldots a}_{n}$





In the handouts is a similar graph for $(a^*)^* \cdot b$ and Java 8, JavaScript and Python.

(Basic) Regular Expressions

Their inductive definition:

r	$::=$	$\mathbf{0}$	nothing
	$\mid$	$\mathbf{1}$	empty string / "" / []
	$\mid$	c	character
	$\mid$	$r_1 + r_2$	alternative / choice
	$\mid$	$r_1 \cdot r_2$	sequence
	$\mid$	r^*	star (zero or more)

When Are Two Regular Expressions Equivalent?

Two regular expressions r_1 and r_2 are **equivalent** provided:

$$r_1 \equiv r_2 \quad \stackrel{\text{def}}{=} \quad L(r_1) = L(r_2)$$

Some Concrete Equivalences

$$
\begin{array}{rcl}
(a+b)+c & \equiv & a+(b+c)\\
a+a & \equiv & a\\
a+b & \equiv & b+a\\
(a\cdot b)\cdot c & \equiv & a\cdot(b\cdot c)\\
c\cdot(a+b) & \equiv & (c\cdot a)+(c\cdot b)
\end{array}
$$

Some Concrete Equivalences

$$
\begin{array}{rcl}
(a+b)+c & \equiv & a+(b+c) \\
a+a & \equiv & a \\
a+b & \equiv & b+a \\
(a\cdot b)\cdot c & \equiv & a\cdot(b\cdot c) \\
c\cdot(a+b) & \equiv & (c\cdot a)+(c\cdot b)
\end{array}
$$

$$
\begin{array}{rcl}
a\cdot a & \not\equiv & a \\
a+(b\cdot c) & \not\equiv & (a+b)\cdot(a+c)
\end{array}
$$

Some Corner Cases

$$
\begin{aligned}
a \cdot \mathbf{0} &\not\equiv a \\
a + \mathbf{1} &\not\equiv a \\
\mathbf{1} &\equiv \mathbf{0}^* \\
\mathbf{1}^* &\equiv \mathbf{1} \\
\mathbf{0}^* &\not\equiv \mathbf{0}
\end{aligned}
$$

Some Simplification Rules

$$
\begin{array}{rcl}
r + \mathbf{0} & \equiv & r \\
\mathbf{0} + r & \equiv & r \\
r \cdot \mathbf{1} & \equiv & r \\
\mathbf{1} \cdot r & \equiv & r \\
r \cdot \mathbf{0} & \equiv & \mathbf{0} \\
\mathbf{0} \cdot r & \equiv & \mathbf{0} \\
r + r & \equiv & r
\end{array}
$$

Simplification Example

$$((\mathbf{1}\cdot b)+\mathbf{0})\cdot r \;\Rightarrow\; ((\underline{\mathbf{1}\cdot b})+\mathbf{0})\cdot r$$

$$= (\underline{b+\mathbf{0}})\cdot r$$

$$= b\cdot r$$

Simplification Example

$$
\begin{array}{rcl}
((\mathbf{0} \cdot b) + \mathbf{0}) \cdot r & \Rightarrow & ((\underline{\mathbf{0} \cdot b}) + \mathbf{0}) \cdot r \\
& = & (\underline{\mathbf{0} + \mathbf{0}}) \cdot r \\
& = & \mathbf{0} \cdot r \\
& = & \mathbf{0}
\end{array}
$$

Semantic Derivative

- The **Semantic Derivative** of a language w.r.t. to a character c:

$$Der\,c\,A \stackrel{\text{def}}{=} \{s \mid c::s \in A\}$$

For $A = \{foo, bar, frak\}$ then

$$\begin{aligned} Der\,f\,A &= \{oo, rak\} \\ Der\,b\,A &= \{ar\} \\ Der\,a\,A &= \{\} \end{aligned}$$

Semantic Derivative

- The **Semantic Derivative** of a language w.r.t. to a character c:

$$Der\,c\,A \stackrel{\text{def}}{=} \{s \mid c :: s \in A\}$$

For $A = \{foo, bar, frak\}$ then

$$\begin{aligned} Der\,f\,A &= \{oo, rak\} \\ Der\,b\,A &= \{ar\} \\ Der\,a\,A &= \{\} \end{aligned}$$

We can extend this definition to strings

$$Ders\,s\,A = \{s' \mid s @ s' \in A\}$$

The Specification for Matching

A regular expression r matches a string s provided:

$$s \in L(r)$$

...and the point of the this lecture is to decide this problem as fast as possible (unlike Python, Ruby, Java etc)

Brzozowski's Algorithm (1)

...whether a regular expression can match the empty string:

$$
\begin{array}{lcl}
nullable(\mathbf{0}) & \stackrel{\text{def}}{=} & \mathit{false} \\
nullable(\mathbf{1}) & \stackrel{\text{def}}{=} & \mathit{true} \\
nullable(c) & \stackrel{\text{def}}{=} & \mathit{false} \\
nullable(r_1 + r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \vee nullable(r_2) \\
nullable(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \wedge nullable(r_2) \\
nullable(r^*) & \stackrel{\text{def}}{=} & \mathit{true}
\end{array}
$$

The Derivative of a Rexp

If r matches the string $c :: s$, what is a regular expression that matches just s?

$der\ c\ r$ gives the answer, Brzozowski 1964

The Derivative of a Rexp

$$
\begin{array}{lcl}
der\,c\,(\mathbf{0}) & \stackrel{\text{def}}{=} & \mathbf{0}\\
der\,c\,(\mathbf{1}) & \stackrel{\text{def}}{=} & \mathbf{0}\\
der\,c\,(d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \mathbf{1} \text{ else } \mathbf{0}\\
der\,c\,(r_1 + r_2) & \stackrel{\text{def}}{=} & der\,c\,r_1 + der\,c\,r_2\\
der\,c\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } nullable(r_1)\\
 & & \text{then } (der\,c\,r_1) \cdot r_2 + der\,c\,r_2\\
 & & \text{else } (der\,c\,r_1) \cdot r_2\\
der\,c\,(r^*) & \stackrel{\text{def}}{=} & (der\,c\,r) \cdot (r^*)
\end{array}
$$

The Derivative of a Rexp

$$
\begin{array}{lcl}
der\,c\,(\mathbf{0}) & \stackrel{\text{def}}{=} & \mathbf{0}\\
der\,c\,(\mathbf{1}) & \stackrel{\text{def}}{=} & \mathbf{0}\\
der\,c\,(d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \mathbf{1} \text{ else } \mathbf{0}\\
der\,c\,(r_1 + r_2) & \stackrel{\text{def}}{=} & der\,c\,r_1 + der\,c\,r_2\\
der\,c\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } nullable(r_1)\\
 & & \text{then } (der\,c\,r_1) \cdot r_2 + der\,c\,r_2\\
 & & \text{else } (der\,c\,r_1) \cdot r_2\\
der\,c\,(r^*) & \stackrel{\text{def}}{=} & (der\,c\,r) \cdot (r^*)
\end{array}
$$

$$
\begin{array}{lcl}
ders\,[]\,r & \stackrel{\text{def}}{=} & r\\
ders\,(c::s)\,r & \stackrel{\text{def}}{=} & ders\,s\,(der\,c\,r)
\end{array}
$$

Examples

Given $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$ what is

$$der\, a\, r = ?$$
$$der\, b\, r = ?$$
$$der\, c\, r = ?$$

Derivative Example

Given $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$ what is

$$
\begin{aligned}
der\,a\,((a \cdot b) + b)^* \quad &\Rightarrow \quad der\,a\,\underline{((a \cdot b) + b)^*}\\
&= \quad (der\,a\,(\underline{(a \cdot b) + b})) \cdot r\\
&= \quad ((der\,a\,(\underline{a \cdot b})) + (der\,a\,b)) \cdot r\\
&= \quad (((der\,a\,\underline{a}) \cdot b) + (der\,a\,b)) \cdot r\\
&= \quad ((\mathbf{1} \cdot b) + (der\,a\,\underline{b})) \cdot r\\
&= \quad ((\mathbf{1} \cdot b) + \mathbf{0}) \cdot r
\end{aligned}
$$

The Brzozowski Algorithm

$$matcher\ r\ s \stackrel{\text{def}}{=} nullable(ders\ s\ r)$$

Brzozowski: An Example

Does r_1 match abc?

Step 1:	build derivative of a and r_1	$(r_2 = der\ a\ r_1)$
Step 2:	build derivative of b and r_2	$(r_3 = der\ b\ r_2)$
Step 3:	build derivative of c and r_3	$(r_4 = der\ c\ r_3)$
Step 4:	the string is exhausted: test whether r_4 can recognise the empty string	$(nullable(r_4))$
Output:	result of the test $\Rightarrow$ *true* or *false*	

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r_1 then

1. $Der\,a\,(L(r_1))$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r_1 then

1. $Der\, a\, (L(r_1))$
2. $Der\, b\, (Der\, a\, (L(r_1)))$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r_1 then

1. $Der\, a\, (L(r_1))$
2. $Der\, b\, (Der\, a\, (L(r_1)))$
3. $Der\, c\, (Der\, b\, (Der\, a\, (L(r_1))))$

4. finally we test whether the empty string is in this set; same for $Ders\, abc\, (L(r_1))$.

The matching algorithm works similarly, just over regular expressions instead of sets.

The Idea with Derivatives

Input: string abc and regular expression r

1. $der\,a\,r$
2. $der\,b\,(der\,a\,r)$
3. $der\,c\,(der\,b\,(der\,a\,r))$

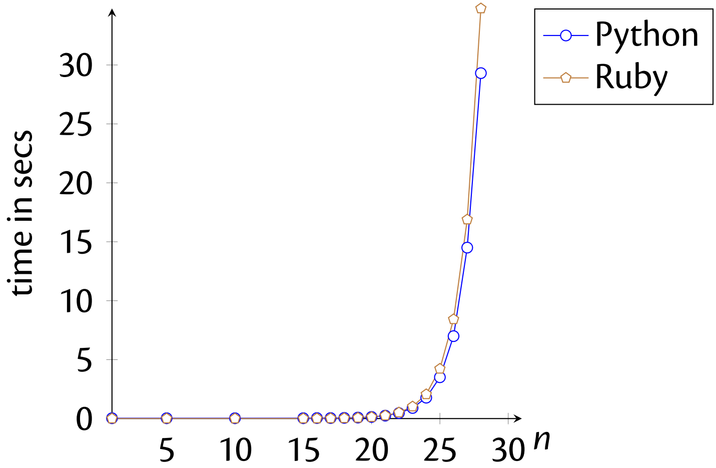
The Idea with Derivatives

Input: string abc and regular expression r

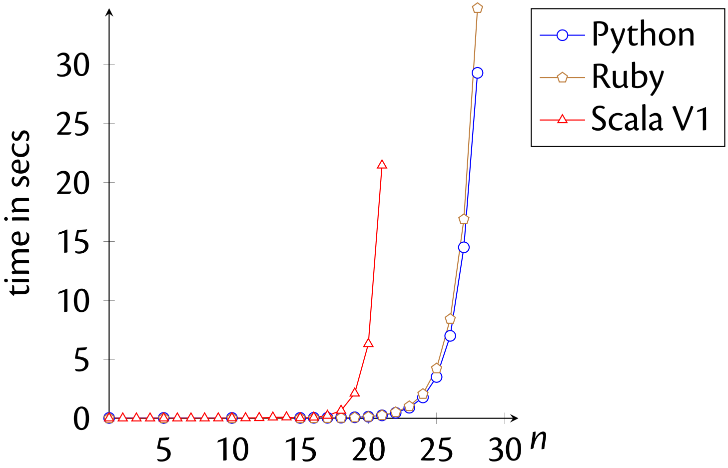
1. $der\, a\, r$
2. $der\, b\, (der\, a\, r)$
3. $der\, c\, (der\, b\, (der\, a\, r))$

4. finally check whether the last regular expression can match the empty string

$$a?^{\{n\}} \cdot a^{\{n\}}$$



Oops... $a^{?\{n\}} \cdot a^{\{n\}}$



A Problem

We represented the "n-times" $a^{\{n\}}$ as a sequence regular expression:

$$
\begin{array}{rl}
0: & \mathbf{1} \\
1: & a \\
2: & a \cdot a \\
3: & a \cdot a \cdot a \\
 & \ldots \\
13: & a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \\
 & \ldots \\
20: &
\end{array}
$$

This problem is aggravated with $a^?$ being represented as $a + \mathbf{1}$.

Solving the Problem

What happens if we extend our regular expressions with explicit constructors

$$
\begin{array}{rcl}
r & ::= & \ldots \\
  & | & r^{\{n\}} \\
  & | & r^{?}
\end{array}
$$

What is their meaning?
What are the cases for *nullable* and *der*?

der for *n*-times

Case $n = 2$ and $r \cdot r$:

$$der\,c\,(r \cdot r) \stackrel{\text{def}}{=} \begin{array}{ll} \text{if} & nullable(r) \\ \text{then} & (der\,c\,r) \cdot r + der\,c\,r \\ \text{else} & (der\,c\,r) \cdot r \end{array}$$

der for *n*-times

Case $n = 2$ and $r \cdot r$:

$$
\begin{array}{lcl}
\textit{der}\, c\, (r \cdot r) & \stackrel{\text{def}}{=} & \text{if}\;\; \textit{nullable}(r) \\
 & & \text{then}\;\; (\textit{der}\, c\, r) \cdot r + \textit{der}\, c\, r \\
 & & \text{else}\;\; (\textit{der}\, c\, r) \cdot r
\end{array}
$$

my claim (in this case) $\equiv (\textit{der}\, c\, r) \cdot r$

der for n-times

Case $n = 2$ and $r \cdot r$:

$$der\,c\,(r \cdot r) \stackrel{\text{def}}{=} \begin{array}{ll} \text{if} & nullable(r) \\ \text{then} & (der\,c\,r) \cdot r + der\,c\,r \\ \text{else} & (der\,c\,r) \cdot r \end{array}$$

my claim (in this case) $\equiv (der\,c\,r) \cdot r$

We know $nullable(r)$ holds!

We know $nullable(r)$ holds!

$$(der\,c\,r) \cdot r + der\,c\,r$$

We know $nullable(r)$ holds!

$$(der\,c\,r) \cdot r + der\,c\,r \quad \equiv \quad (der\,c\,r) \cdot r + (der\,c\,r) \cdot \mathbf{1}$$

We know $\mathit{nullable}(r)$ holds!

$$
\begin{aligned}
(\mathit{der}\,c\,r)\cdot r + \mathit{der}\,c\,r &\equiv (\mathit{der}\,c\,r)\cdot r + (\mathit{der}\,c\,r)\cdot \mathbf{1}\\
&\equiv (\mathit{der}\,c\,r)\cdot (r + \mathbf{1})
\end{aligned}
$$

We know $nullable(r)$ holds!

$$
\begin{aligned}
(der\,c\,r)\cdot r + der\,c\,r &\equiv (der\,c\,r)\cdot r + (der\,c\,r)\cdot \mathbf{1}\\
&\equiv (der\,c\,r)\cdot (r + \mathbf{1})\\
&\equiv (der\,c\,r)\cdot r
\end{aligned}
$$

(remember r is nullable)

We know $\mathit{nullable}(r)$ holds!

$$
\begin{aligned}
(\mathit{der}\,c\,r)\cdot r + \mathit{der}\,c\,r \quad &\equiv \quad (\mathit{der}\,c\,r)\cdot r + (\mathit{der}\,c\,r)\cdot \mathbf{1}\\
&\equiv \quad (\mathit{der}\,c\,r)\cdot (r + \mathbf{1})\\
&\equiv \quad (\mathit{der}\,c\,r)\cdot r
\end{aligned}
$$

(remember r is nullable)

$$
\mathit{der}\,c\,(r\cdot r) \stackrel{\text{def}}{=} \begin{array}{l}\text{if } \mathit{nullable}(r)\\ \text{then } (\mathit{der}\,c\,r)\cdot r + \mathit{der}\,c\,r\\ \text{else } (\mathit{der}\,c\,r)\cdot r\end{array}
$$

We know $nullable(r)$ holds!

$$
\begin{aligned}
(der\,c\,r)\cdot r + der\,c\,r &\equiv (der\,c\,r)\cdot r + (der\,c\,r)\cdot \mathbf{1}\\
&\equiv (der\,c\,r)\cdot (r + \mathbf{1})\\
&\equiv (der\,c\,r)\cdot r
\end{aligned}
$$

(remember r is nullable)

$$
der\,c\,(r\cdot r) \stackrel{\text{def}}{=} \begin{array}{l}\text{if } nullable(r)\\ \text{then } (der\,c\,r)\cdot r\\ \text{else } (der\,c\,r)\cdot r\end{array}
$$

We know $nullable(r)$ holds!

$$
\begin{aligned}
(der\,c\,r) \cdot r + der\,c\,r \quad &\equiv \quad (der\,c\,r) \cdot r + (der\,c\,r) \cdot \mathbf{1} \\
&\equiv \quad (der\,c\,r) \cdot (r + \mathbf{1}) \\
&\equiv \quad (der\,c\,r) \cdot r
\end{aligned}
$$

(remember r is nullable)

$$der\,c\,(r \cdot r) \quad \stackrel{\text{def}}{=} \quad (der\,c\,r) \cdot r$$

	$r\{n\}$	der
$n = 0:$	$\mathbf{1}$	$\mathbf{0}$
$n = 1:$	r	$(der\,c\,r)$
$n = 2:$	$r \cdot r$	$(der\,c\,r) \cdot r$
$n = 3:$	$r \cdot r \cdot r$	???
$\vdots$		

	$r\{n\}$	der
$n = 0:$	$\mathbf{1}$	$\mathbf{0}$
$n = 1:$	r	$(der\, c\, r)$
$n = 2:$	$r \cdot r$	$(der\, c\, r) \cdot r$
$n = 3:$	$r \cdot r \cdot r$	$(der\, c\, r) \cdot r \cdot r$
$\vdots$		

	$r\{n\}$	der
$n = 0:$	$\mathbf{1}$	$\mathbf{0}$
$n = 1:$	r	$(der\,c\,r)$
$n = 2:$	$r \cdot r$	$(der\,c\,r) \cdot r$
$n = 3:$	$r \cdot r \cdot r$	$(der\,c\,r) \cdot r \cdot r$
$\vdots$		

$$nullable(r\{n\}) \stackrel{\text{def}}{=} \text{if } n = 0 \text{ then } true \text{ else } nullable(r)$$

$$der\,c\,(r\{n\}) \stackrel{\text{def}}{=} \text{if } n = 0 \text{ then } \mathbf{0} \text{ else } (der\,c\,r) \cdot r\{n-1\}$$

$$
\begin{array}{lcl}
der\,c\,(r \cdot r \cdot r) & \stackrel{\text{def}}{=} & \text{if}\; nullable(r) \\
 & & \text{then}\;\; (der\,c\,r) \cdot r \cdot r + der\,c\,(r \cdot r) \\
 & & \text{else}\;\; (der\,c\,r) \cdot r \cdot r
\end{array}
$$

$$
\begin{aligned}
der\,c\,(r\cdot r\cdot r) \;\stackrel{\text{def}}{=}\;& \text{if } nullable(r)\\
& \text{then }\; (der\,c\,r)\cdot r\cdot r + (der\,c\,r)\cdot r\\
& \text{else }\; (der\,c\,r)\cdot r\cdot r
\end{aligned}
$$

$$
\begin{aligned}
der\,c\,(r\cdot r\cdot r) \;\stackrel{\text{def}}{=}\;\; & \text{if}\; nullable(r)\\
& \text{then}\;\; (der\,c\,r)\cdot(r\cdot r + r)\\
& \text{else}\;\; (der\,c\,r)\cdot r\cdot r
\end{aligned}
$$

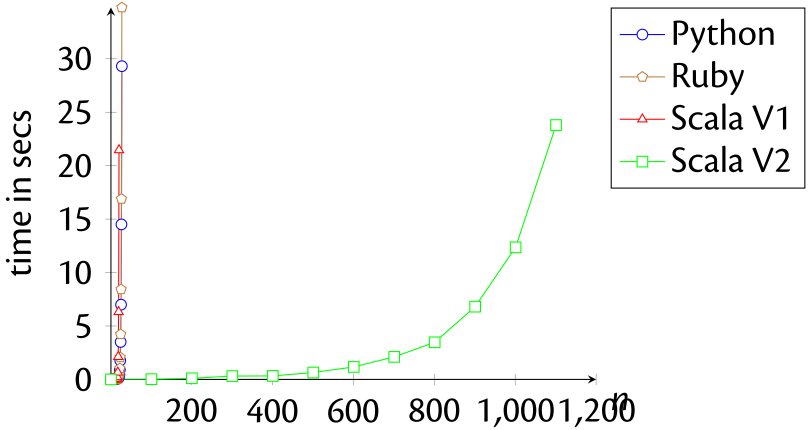
$$
\begin{aligned}
der\,c\,(r\cdot r\cdot r) \;\stackrel{\text{def}}{=}\;& \text{if } nullable(r)\\
& \text{then }\;(der\,c\,r)\cdot(r\cdot(r+\mathbf{1}))\\
& \text{else }\;(der\,c\,r)\cdot r\cdot r
\end{aligned}
$$

$$
\begin{array}{rcl}
der\,c\,(r \cdot r \cdot r) & \stackrel{\text{def}}{=} & \text{if } nullable(r) \\
 & & \text{then } \; (der\,c\,r) \cdot (r \cdot r) \\
 & & \text{else } \; (der\,c\,r) \cdot r \cdot r
\end{array}
$$

$$
\begin{array}{lcl}
der\,c\,(r \cdot r \cdot r) & \stackrel{\text{def}}{=} & \text{if}\; nullable(r) \\
& & \text{then}\;\; (der\,c\,r) \cdot r \cdot r \\
& & \text{else}\;\; (der\,c\,r) \cdot r \cdot r
\end{array}
$$

$$der\,c\,(r \cdot r \cdot r) \stackrel{\text{def}}{=} (der\,c\,r) \cdot r \cdot r$$

Brzozowski: $a^{?\{n\}} \cdot a^{\{n\}}$



Examples

Recall the example of $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$ with

$$\begin{aligned} der\,a\,r &= ((\mathbf{1} \cdot b) + \mathbf{0}) \cdot r \\ der\,b\,r &= ((\mathbf{0} \cdot b) + \mathbf{1}) \cdot r \\ der\,c\,r &= ((\mathbf{0} \cdot b) + \mathbf{0}) \cdot r \end{aligned}$$

What are these regular expressions equivalent to?

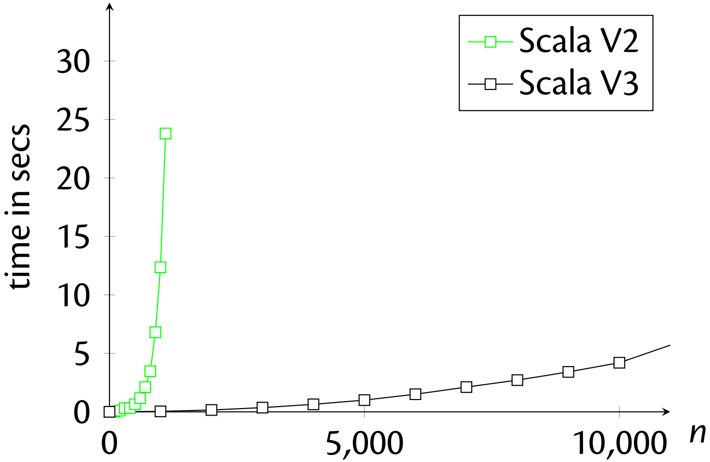
Simplification Rules

$$
\begin{array}{rcl}
r + \mathbf{0} & \Rightarrow & r \\
\mathbf{0} + r & \Rightarrow & r \\
r \cdot \mathbf{1} & \Rightarrow & r \\
\mathbf{1} \cdot r & \Rightarrow & r \\
r \cdot \mathbf{0} & \Rightarrow & \mathbf{0} \\
\mathbf{0} \cdot r & \Rightarrow & \mathbf{0} \\
r + r & \Rightarrow & r
\end{array}
$$

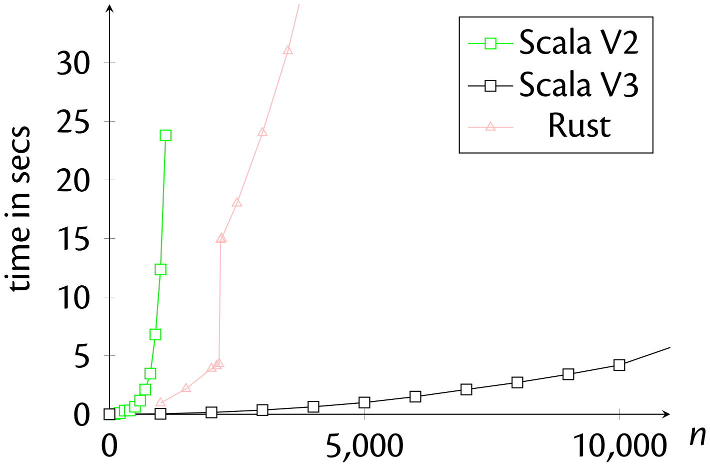
```scala
def ders(s: List[Char], r: Rexp) : Rexp = s match {
  case Nil => r
  case c::s => ders(s, simp(der(c, r)))
}
```

```scala
def simp(r: Rexp) : Rexp = r match {
  case ALT(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, r2s) => r2s
      case (r1s, ZERO) => r1s
      case (r1s, r2s) =>
        if (r1s == r2s) r1s else ALT(r1s, r2s)
    }
  }
  case SEQ(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, _) => ZERO
      case (_, ZERO) => ZERO
      case (ONE, r2s) => r2s
      case (r1s, ONE) => r1s
      case (r1s, r2s) => SEQ(r1s, r2s)
    }
  }
  case r => r
}
```

Brzozowski: $a^{?\{n\}} \cdot a^{\{n\}}$

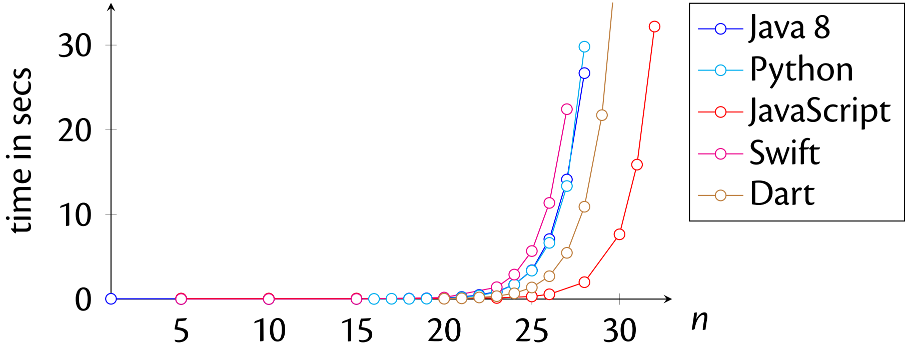


Brzozowski: $a^{?\{n\}} \cdot a^{\{n\}}$



code by Archie Collard

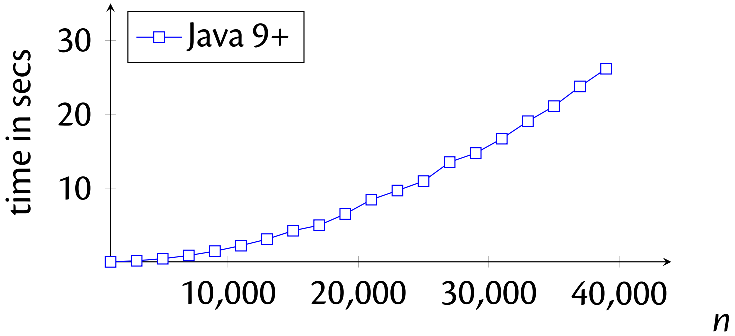
Another Example $(a^*)^* \cdot b$



Regex: $(a^*)^* \cdot b$

Strings of the form $\underbrace{a \ldots a}_{n}$

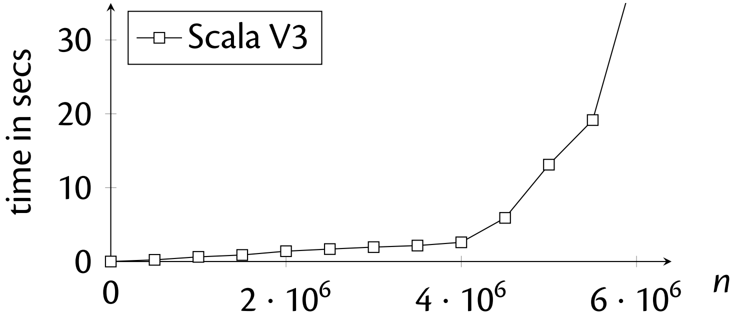
Same Example in Java 9+



Regex: $(a^*)^* \cdot b$

Strings of the form $\underbrace{a \ldots a}_{n}$

...and with Brzozowski



Regex: $(a^*)^* \cdot b$

Strings of the form $\underbrace{a \ldots a}_{n}$

What is good about this Alg.

- extends to most regular expressions, for example $\sim r$ (next slide)
- is easy to implement in a functional language
- the algorithm is already quite old; there is still work to be done to use it as a tokenizer (that is relatively new work)
- we can prove its correctness...(another video)

Negation of Regular Expr's

- $\sim r$ (everything that r cannot recognise)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} not\,(nullable(r))$
- $der\,c\,(\sim r) \stackrel{\text{def}}{=} \sim (der\,c\,r)$

Negation of Regular Expr's

- $\sim r$ (everything that r cannot recognise)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} not\,(nullable(r))$
- $der\,c\,(\sim r) \stackrel{\text{def}}{=} \sim (der\,c\,r)$

Used often for recognising comments:

$$/ \cdot * \cdot (\sim ([a\text{-}z]^* \cdot * \cdot / \cdot [a\text{-}z]^*)) \cdot * \cdot /$$

The Specification for Matching

A regular expression r matches a string s provided:

$$s \in L(r)$$

$matcher\, s\, r$ if and only if $s \in L(r)$

The Specification for Matching

A regular expression r matches a string s provided:

$$s \in L(r)$$

$$\forall\, r\, s.\;\; matcher\, s\, r \;\text{ if and only if }\; s \in L(r)$$

nullable and *der*

The central properties:

$$nullable(r) \;\; \textit{if and only if} \;\; [] \in L(r)$$

nullable and *der*

The central properties:

$$nullable(r) \;\; \textit{if and only if} \;\; [] \in L(r)$$

$$L(der\,c\,r) \;=\; Der\,c\,(L(r))$$

nullable and *der*

The central properties:

$$\forall r.\quad nullable(r)\ \textit{if and only if}\ [] \in L(r)$$

$$\forall r\, c.\ L(der\, c\, r) \;=\; Der\, c\, (L(r))$$

Proofs about Rexps

Remember their inductive definition:

$$
\begin{array}{rcl}
r & ::= & \mathbf{0} \\
  & \mid & \mathbf{1} \\
  & \mid & c \\
  & \mid & r_1 \cdot r_2 \\
  & \mid & r_1 + r_2 \\
  & \mid & r^*
\end{array}
$$

If we want to prove something, say a property $P(r)$, for all regular expressions r then …

Proofs about Rexp (2)

- P holds for $\mathbf{0}$, $\mathbf{1}$ and c
- P holds for $r_1 + r_2$ under the assumption that P already holds for r_1 and r_2.
- P holds for $r_1 \cdot r_2$ under the assumption that P already holds for r_1 and r_2.
- P holds for r^* under the assumption that P already holds for r.

Proofs about Rexp

Assume $P(r)$ is the property:

$$nullable(r) \text{ if and only if } [] \in L(r)$$